

分布式系统 Lab1 设计文档

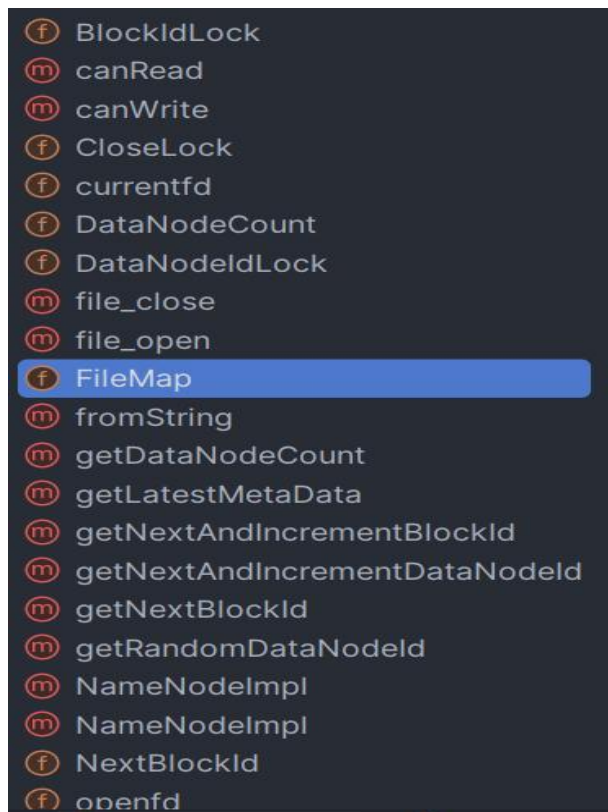
1. FSImage 的设计

在代码的源文件夹下面添加了 FSImage 文件夹保存 FSImage 文件。它保存了 NameNodeImpl 中的所有信息，包括 DataNode 开启的数量，每个 DataNode 接下来可分配的 Random 的 BlockId，文件名和 metadata 间的关系，当前 fd 的值，并且可由此得到下一个 fd 值。

2. 文件和数据块的映射和数据块定位

文件和数据块的映射关系通过在 NameNodeImpl 中的 Map<String, MetaDataInfo>体现。MetaDataInfo 类中的 BlockInfo 储存了数据块的信息，包括 DataNodeId 和 BlockId。一个文件可以通过多个数据块去储存，每个数据块可以通过 DataNodeId 确定。由于数据块是按序保存的，所以在 BlockInfo 中的 index 就意味着它是组成这个文件的第几个数据块。

3. NameNodeImpl 的实现



在 NameNodeImpl 中，实现了包括文件的打开、关闭、权限检查、元数据更新等操作，并提供了磁盘持久化和恢复。我在每个主要函数前面都写了注释，标明了每个函数的作用。

NameNodeImpl 的元数据操作方法：

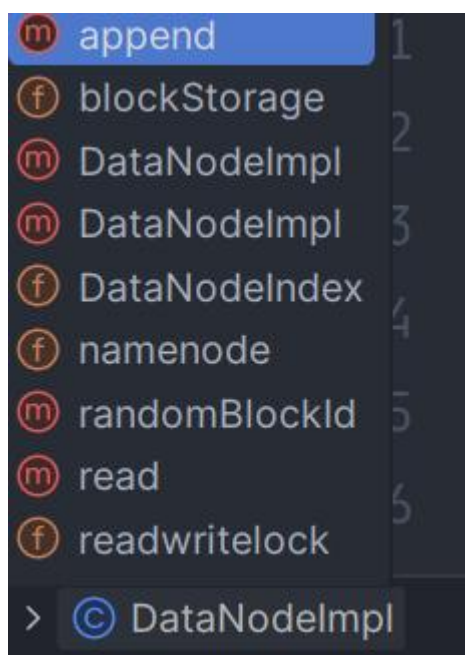
- getNextAndIncrementDataNodeId()：获取下一个数据节点的 ID，并增加数据节点计数。
- getNextBlockId(int dataNodeId)：获取指定数据节点的下一个块 ID。
- getNextAndIncrementBlockId(int DataNodeId)：获取指定数据节点的下一个块 ID 并增加块 ID 计数。
- getRandomDataNodeId()：根据已有的数据节点数量，产生一个随机的数据节点 ID。

`Open (String filepath, int mode)` 打开文件并返回文件元数据信息的字符串表示。如果文件存在，会将 `metadata, mode` 保存在 `FileDesc` 中。如果文件不存在，那么会创建该文件对应的 `metadata`。

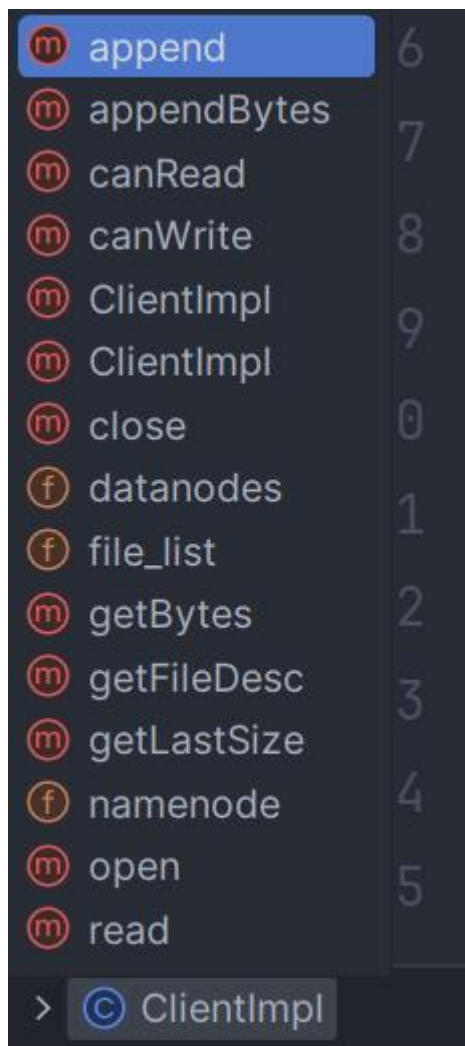
`Close (String fileinfo)`，会把 `FileDesc` 的内容变成 `NameNodeImpl` 的对象并且传递给 `NameNodeImpl`，它会更新 `metadata` 和 `FSImage`。

4. `DataNodeImpl` 的实现

首先通过 CORBA 注册 `DataNode` 服务，并在命名服务中绑定 `DataNode` 引用，初始化 `DataNode` 的索引，然后通过文件输入流读取指定数据块的内容，并将其写入字节数组中，然后返回字节数组。其中 `append(int block_id, byte[] bytes)` 函数是通过文件输出流将字节数组追加到指定数据块的末尾，最后生成一个下一个可用的块 ID。



5. ClientImpl 的实现



- `ClientImpl` 中实现了客户端的基本功能，包括打开文件、读取文件、追加数据到文件末尾和关闭文件等操作。首先通过指定的 ORB 参数，获取 `NameNode` 和 `DataNode` 引用，初始化客户端。在文件读写中，`open(String filepath, int mode)`: 打开文件并获取元数据信息，返回文件描述符的 ID。`append(int fd, byte[] bytes)`: 在指定文件描述符的文件末尾追加字节数组，并更新元数据信息，包括修改时间戳，文件大小等。`read(int fd)`: 读取文件内容并返回字节数组。在工具方法中，各个对应的函数会判断 `fd` 的

权限并作出相应的读写操作，或者是根据 `fd` 返回对应的文件描述对象在 `appendBytes` 方法中，如果数据量大于一个数据块的大小，会拆分字节数组并追加到多个数据块中，保证文件的完整性和存储。

6. ClientNodeLauncher 实现命令解析

该节点与分布式文件系统的 `ClientImpl` 进行交互，创建 `ClientImpl` 类的实例作为客户端的具体实现，创建 `Scanner` 用于接收用户输入的命令。接着解析用户输入的命令，并根据命令调用相应的 `ClientImpl` 方法。

`open`: 打开文件，根据用户输入的文件路径和模式 `mode` 调用 `ClientImpl` 的 `open` 方法。

`read`: 读取文件，根据用户输入的 `fd` 调用 `ClientImpl` 的 `read` 方法。

`append`: 追加数据到文件末尾，根据用户输入的数据节点 ID 和数据调用 `ClientImpl` 的 `append` 方法。

`close`: 关闭文件，根据用户输入的文件描述符调用 `ClientImpl` 的 `close` 方法。

`exit`: 退出程序。





7. 并发性以及锁的实现

由于本次 lab 需要对文件实现权限管理，即每次只能够有一个线程去写文件，则它在写的时候便会拿到这个锁，因为写文件会创建新的数据块，那么这个数据块对于读来说就不可访问，只有在

close 操作之后才可以通过 read 访问修改或者添加的 metadata。同样，对于 read 操作而言，其会从 NameNodeImpl 中获取 latest metadata，close 会修改 metadata，那么这里也需要锁实现互斥化。最后，在 DataNode 开启和分配数据块的时候，需要保证这些的 ID 都是唯一的，因此也需要锁去实现。

自动测试结果与手动测试

1.自动测试全过

```
1    
2   public class DataNodeTest {  
  
1   public class NameNodeTest {  
  
1  
2   public class ClientTest {  
14 using
```

2.手动测试

```
NameNode is obtained
DataNode0 is obtained
DataNode1 is obtained
Enter a command:
open xqc.txt w
INFO: fd=1
Enter a command:
append 1 hello world
INFO: write done
Enter a command:
read 1
Cannot Read!
Enter a command:
close 1
INFO: fd 1 closed
```

```
close 1
INFO: fd 1 closed
Enter a command:
open xqc.txt rw
INFO: fd=2
Enter a command:
read 2
hello world
Enter a command:
exit
INFO: bye
```

